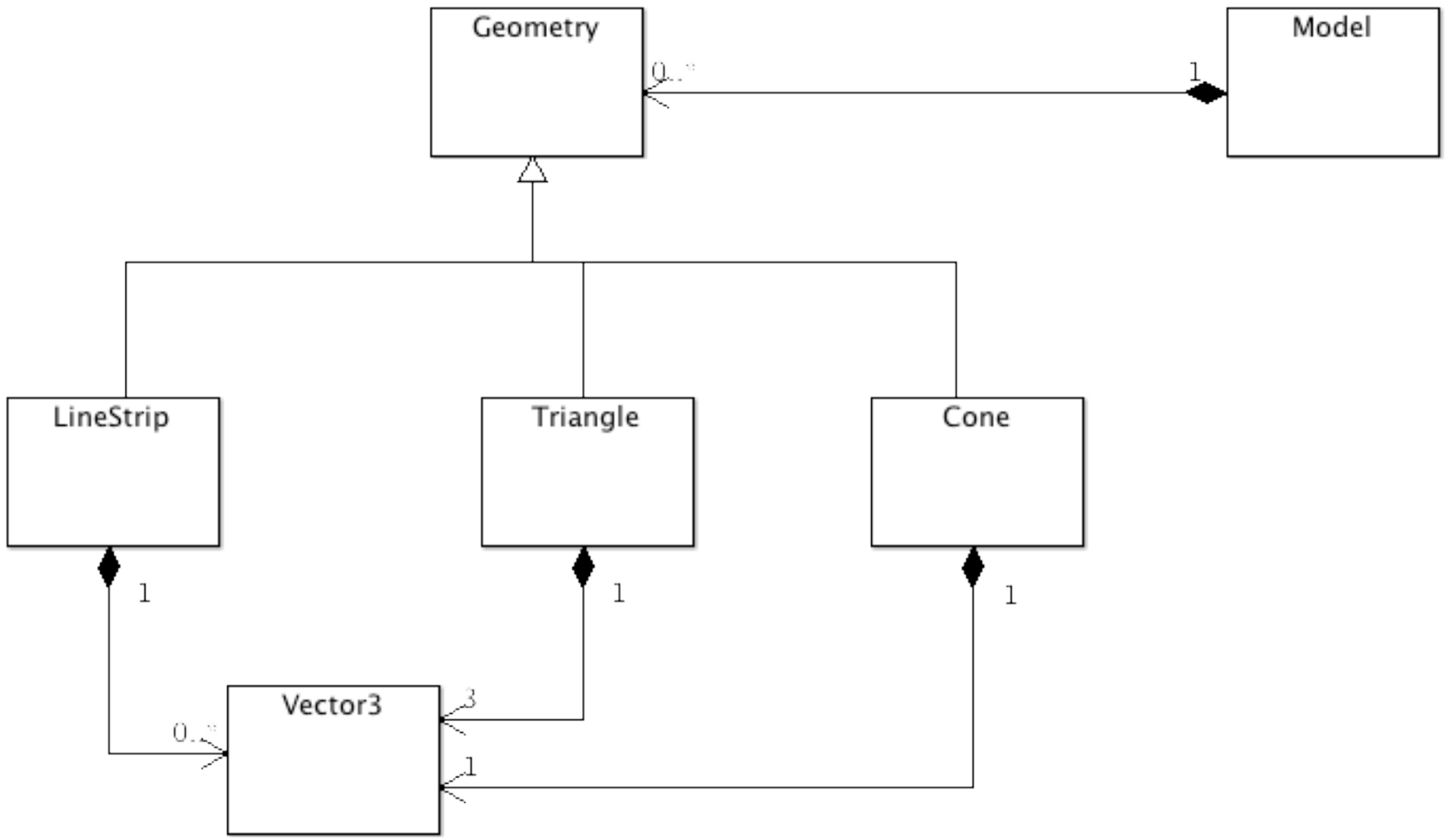
Assignment 1 Shell Code

OpenGL

Design



Vector3

- A floating point three-dimensional vector.

- Can be used either as a traditional vector (indicating a direction) or a Vertex (treating X, Y, and Z as coordinates).

- Static variable represent Unit Vectors (or directions) on X, Y and Z axis.

```
struct Vector3
{
  float X;
  float Y;
  float Z;

  static Vector3 UnitX;
  static Vector3 UnitY;
  static Vector3 UnitZ;

  Vector3(float x, float y, float z);
  Vector3(float value);
  Vector3();
  Vector3(std::istream& is);

  void render();
};
```

Geometry

- Abstract class
- Represents an entity that can be rendered
- Has a unique name

```
struct Geometry
{
  Geometry(std::istream& is);

  virtual void render()=0;

  std::string name;
};
```

LineStrip

- A Sequence of Vector3s, which will be rendered as a GL_LINE_STRIP

```
struct LineStrip : public Geometry
{
  std::vector<Vector3> vertices;

  LineStrip(std::istream& is);
  void render();
};
```

Triangle

- Three Vector3's, rendered as a GL_TRIANGLES

```
struct Triangle : public Geometry
{
  Vector3 p1, p2, p3;

  Triangle(std::istream& is);
  void render();
};
```

Cone

- ?

```
struct Cone : public Geometry
{

};
```

Model

- Loads geometry from a file.
- Store each geometric entity *by name* in a map

```
typedef std::map <std::string, Geometry*>  GeometryMap;
typedef GeometryMap::iterator GeometryMapIterator;

struct Model
{
  int minX, maxX, minY, maxY, minZ, maxZ;
  GeometryMap entities;

  Geometry* get(std::string name);
  Model(std::istream& is);
  ~Model();
  void render();
};
```

Model - Maps:

```
typedef std::map <std::string, Geometry*>  GeometryMap;
typedef GeometryMap::iterator GeometryMapIterator;

struct Model
{
  //
  GeometryMap entities;

  Geometry* get(std::string name);
  //
};
```

- **GeometryMap** is an alias (another name) for a map of
  - String -> Geometry Pointer
  - I.e. A GeometryMap is a table of name/value pairs, where the name is a string and the value is a pointer to some Geometry object
- **GeometryMapIterator** is an alias for a type which can iterate through a GeometryMap
- get(), when called on a model, will retrieve a named geometry object from the loaded model.

Model Loading

- Each Geometric Entity is loaded by its own constructor.
- This includes loading the name of the entity.
- All entities are then stored in a map, keyed on the name.

```
Model::Model(istream &is)
{
  int size;
  skipComment(is);
  is >> minX >> maxX >> minY >> maxY >> minZ >> maxZ;
  skipComment(is);
  is >> size;
  for (int i = 0; i < size; i++)
  {
    int typeId;
    skipComment(is);
    is >> typeId;
    Geometry *entity=0;
    switch (typeId)
    {
      case LineStripId: {
                          entity = new LineStrip(is);
                          break;
                        }
      case TriangleId:  {
                          entity = new Triangle(is);
                          break;
                        }
    }
    if (entity != 0)
    {
        entities[entity->name] = entity;
    }
  }
}
```

Model: Retrieving an Entity

- find() method returns an iterator.

- If the iterator != end(), then a match has been found in the map, and find returns an iterator for this entry.

- The iterator has two methods -

  - first() is the key (the name of the entity)

  - second() is the entity itself - a Geometry* in this case

```
Geometry* Model::get(string name)
{
  if (entities.find(name) != entities.end())
  {
    return entities.find(name)->second;
  }
  return 0;
}
```

Model File Format

- Not included in archive!
- Note that each entity has a name.
- This will be the key used by the model
- The model can be queried for any of these names, and it will return the appropriate Geometry* object

```
#scale
-300 300 -300 300 -300 300
#number of entities
3
# 1st entity
0
#name
line1
# number of vertices
2
0 -300  0
0  300  0
# 2nd entity
0
#name
line2
# number of vertices
2
-300 0  0
300  0  0
# 3rd entity
2
#name
cone1
# origin
150 150 50
# radius
50
```

Main

- Load a model
- Render a model

```
Model *model;

Model* loadModel(const char *filename)
{
  //...
}

void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  model->render();
  glFlush();
}

void setupRC()
{
  //...
  model = loadModel("model.txt");
  if (model != 0)
  {
    glOrtho(model->minX, model->maxX,
            model->minY, model->maxY, model->minZ, model->maxZ);
  }
}

int main(int argc, char* argv[])
{
  //...
  if (model != 0)
  {
    glutMainLoop();
  }
  //...
  return 0;
}
```

Querying a Model for an Entity

- model->get will return a specific entity.

- If you know for certain that is is a Cone for instance, it can be downcasted to a Cone*, and then used as you wish.

```
Geometry *thing = model->get("triangle1");

Geometry *object = model->get("cone1");


Cone* cone = (Cone*) object;
```